METHODS AND SYSTEMS FOR USING DISTRIBUTED BUSINESS DATA USING OBSERVATION TECHNOLOGY TO AVOID THE NEED TO INTEGRATE SERVERS AND CLIENTS

FIELD OF THE INVENTION

[0001] The present invention relates to information systems (e.g., evolving enterprise systems), and more particularly to data management, data analysis, knowledge creation, and decision support.

BACKGROUND OF THE INVENTION

[0002] The ability to act quickly and decisively in today's increasingly competitive marketplace is critical to the success of any organization. As a basis for intelligent decision-making, data is an organization's biggest potential asset; however, to realize its potential, businesses must be able to gather and reconcile data in different formats and from various sources ("data management"); transform that data into knowledge via intelligent analysis ("knowledge creation"); and use that knowledge, in as close to "real time" as possible, as the basis for modifying and optimizing business operations ("decision support").

[0003] Current approaches to data management, analysis, knowledge creation, and decision support focus on the data warehouse and related technologies. Data warehouses, relational databases, and data marts provide the central location where a reconciled version of data extracted from a wide variety of operational systems may be stored. A data warehouse is an informational database that stores and often replicates shareable data from one or more operational databases of record; it supports informational processing by providing a foundation of integrated, enterprise-wide, non-volatile historical data. A data mart may be considered a type of data warehouse that focuses on a particular business segment.

[0004] The challenge of leveraging data via knowledge creation into intelligent business decision-making is formidable and presents difficulties at every step of the process. Current technologies fail to overcome these difficulties for reasons described below.

[0005] Referring to FIG.1 there is shown a schematic diagram depicting a representative enterprise information system of the prior art. Customer 10's and client 1's interaction data sources are a heterogeneous combination of touchpoint-specific, type-specific, platform-specific entities, distributed across the client's enterprise (and usually also geographically). This environment is characterized by:

- Multiple touchpoints (Branch Office 3, Call Center 4, Web Site 5, ATM Machine 6, 9), for both customer information 11 and transactional interaction 12a, 12b
- The various touchpoints are accessed via various mechanisms, both human and machine-based, e.g., walking to branch 3; using the telephone 7 to access call center 4; or using PC 8 to access the Internet
- Each touchpoint is a data source of potentially various types (Relational Database 12b, flat file 12a, Application 12, etc.)
- Each data source type can reside on a particular platform typically a server computer such as IBM DB2/Unix, SQLServer/NT, etc.
- The data from multiple touch points (3,4,5,6) are communicated via a communication network (intranet or internet) 13 to the central information-processing center 2, where it is stored and possibly "mined" to glean any relevant intelligence, which is then fed to the client's headquarters 1.

[0006] Customer interaction data is interactive and "full duplex," meaning that both informational (web content, promotions, etc) and transactional data (account information, etc.) are put into—and in other cases retrieved from—the data sources. However, significantly more data is put into the systems than is retrieved from them because the act of retrieval is itself recorded and stored. In effect, from a data collection and recording perspective, this interaction is a continuously growing "collection" of data.

[0007] Once the data is stored in the Central Processing Center 2, businesses want to benefit from any intelligence derived from the interaction, so a whole host of applications emerge to facilitate transformation of this continuously collected data into useful business intelligence, usually in a batch mode (nightly, monthly, etc.). This process is loosely termed "mining," hence the applications are typically called "data mining" applications.

The basic function of these applications is to "extract" intelligence from past data. This extraction process is represented as an arrow in Fig 1 from Center 2 to headquarter 1.

[0008] Once mined, the business intelligence uses Business Units 1a and 1b to create or modify business rules that form the basis of various decisions typically distributed across several functional areas of the financial enterprise. This process of "pushing" out the intelligence to the various operations decision points is known as "operationalizing" intelligence. Several applications operate in this context, all "using" this intelligence to positively "impact" the interacting customers and possibly increase the value of future interactions. This is also shown in thick arrows in Fig 1 flowing outward from the headquarter 1.

[0009] There are a number of drawbacks of the foregoing prior art as identified below

Data Management

[0010] The first difficulty has to do with *data management*. The quantity of data available to organizations is increasing all the time and originates in diverse, widely distributed, often incompatible—and sometimes unknown—enterprise information sources. As a result, collecting, transforming, integrating, and reconciling all this data (i.e. getting it to the point where an enterprise can even *begin* to make sense of it) is a cumbersome, time-consuming, and extremely expensive process.

[0011] Under the traditional approach, data is transferred into data warehouses and thence to data marts for decision support systems to work with. This presents a few immediate problems:

- As it is not known what data will subsequently be needed, almost all data generated must be warehoused, causing a lot of logistical overhead in storage, backup, and availability of the data assets
- The problem of warehousing from diverse data sources presents a formidable integration challenge

- Once the data warehousing is complete, a change or evolution in an application's data needs to be re-warehoused for the additional information, so the process never finishes
- Worst of all, most of the steps towards making *even this* warehoused data available remain unsolved (though some progress has been made in "right transformation"—i.e., the reorganization, reformatting, and aggregation of *some* data).

Data Analysis/Knowledge Creation

[0012] Current approaches to data analysis have been developed to retrieve selected information from data warehouses. One such is known as an on-line analytical processing system (OLAP). In general, OLAP systems analyze the data from a number of different perspectives and generate reports based on complex analyses against large input data sets.

[0013] Problems with this method include the following:

- **Analytical tools are overwhelmed by quantity of data**
  The analytical tools designed to effect analysis are overwhelmed by the sheer quantity of data they have to work with. Further, because they fail to discriminate adequately between relevant and irrelevant data, these technologies are not especially effective at converting it quickly and intelligently into knowledge. (And as the quantity of data collected increases, as it inevitably does, this problem grows).
- **Method is not suited to exploratory, non-scheduled analysis**
  In order to do "exploratory," non-scheduled, ad hoc analysis (i.e. the kind that creates new knowledge), the user typically engages in a process that involves a step-by-step exploration. At any given step in this process, only a few next steps may be known in advance in terms of the type, kind, format, and detail of the data required to do the analysis at that step. Thus the entire path is usually not fully charted, nor is the entire set of the data required (and its associated parameters like format, depth, type, etc.) clear.

So, the typical process of exploratory data analysis constantly engages the data source in "ad-hoc" queries that get refined as the investigator explores the problem. These "ad-hoc" queries create massive overhead on the system (both organizational structure and the IT systems) and, due to resource constraints, there are delays at every step of the way—delays mostly associated with systems, people, organizations, and their complex inter-dynamics, and having nothing to do with the investigator or the problem at hand. Hence the creation of new knowledge is greatly inhibited. Intuition (usually the starting point of this exploration) is typically diluted if not entirely lost, and the cost implications for the business enterprise are extremely large.

Decision Support

[0014] Decision Support Systems (DSS) technology allows for the use of a "decision tree"-based analysis whereby decisions, uncertainties, events (both known and unknown), risks, and rewards can be captured in a causal, related network to assess the impact on decision making of any change in policy. Such systems range from simple diagrams to vast parallel computational systems. However, the fundamental element missing in DSS technology today is the link or relationships with the "operational" systems. These relationships would ideally facilitate a "fully duplex" communication flow i.e., decision making (via DSS) can get accurate inputs (rather than estimates) from operational systems and conversely, the outputs of DSS can be input directives/objectives of operational systems, rather than just rough manual guidelines to management. This discussion includes feedback of customer responses to offers made to them via such decisions. This feedback, in this "full-duplex" analogy can be used to refine both the rules and the input parameters of the DSS for the next iteration of decisioning - which could be as soon as the next customer in the queue.

[0015] Problems with this method include the following:

- **Batch processing**

Since it is impossible to move such large volumes of data in near-real time, the approach has been to run batch processes or dedicated data extraction utilities during

off-peak hours, the frequency of these processes depending on how urgently the data is needed. This may work for applications where the analytical queries are ad hoc, but for very targeted analytics it works much less well.

- **Functional disjunction between knowledge creation and decision support**
The analysis function (knowledge creation) is not linked at the system level to business operations. Typically, one team conducts analysis and produces reports; another team takes delivery of these reports and makes operation and strategic decisions based on them. There is thus a functional disconnect between knowledge creation and decision-making. This makes coordination difficult and increases the time lag between knowledge discovery and operational decisions—highly problematic in cases where analysis indicates a need for speedy action. A better approach would be one where knowledge creation is linked seamlessly to decision making at the level of the information system via business rules.

- **Decision support is not optimized because not all available data is used**
Current information management systems can take years to implement and are in fact never completed. This flows from the basic fact that all servers must connect with all clients on a one-to-one basis; this requirement means that not all clients are connected to the servers that could supply them with important, if not essential, information. Even if new servers coming online are eventually connected to clients, it is likely that this will not be done in a timely manner. The result is that business decisions are routinely based on incomplete or anachronistic data.

[0016] Architectural Shortcomings

- **No scope for complex feedback loops**
A further shortcoming is that this traditional architecture is built upon the assumption that the triggering response to change the state of the system originates *outside* the system. However, as enterprise applications evolve in sophistication there arises a need for an architecture that can accommodate complex feedback loops (rather than a simple linear flow of information) in the information and knowledge dissemination chain.

- **Unidirectional tools are mismatched with bi-directional data flow**

  Attempts at bridging this bi-directional flow of information and knowledge fall short because they rely on essentially *unidirectional* technologies for data extraction, namely, Data Warehousing, OLAP, and Decision Support System (DSS)-type technologies. This basic conceptual architectural misfit makes the architecture brittle and severely restricts pro-activity and adaptability—essential attributes of next-generation enterprise systems, which will have to adapt to changing intelligence needs.

[0017] These are the main drawbacks with current data management and knowledge creation systems. Finally, although software agents are well known in the art, they have not been used as agents in an evolving or changing enterprise system to observe, notify and filter changes or observations to the various underlying databases.

SUMMARY OF THE INVENTION

[0018] An object of the invention is to overcome these and other drawbacks in existing systems.

[0019] The present invention optimizes the performance of information systems by enabling enterprises to use all available data *without having to aggregate and analyze it,* thus improving the quality and timeliness of knowledge, reducing overhead, and closing the gap between knowledge creation and decision support.

[0020] The present invention is an agent-based "observation management platform." This platform provides a technology infrastructure to rapidly and *proactively* integrate enterprise data (in the form of "observations") from diverse, widely distributed information systems within and across an enterprise and its supply chain partners to sophisticated back-end applications to derive relevant business intelligence. This marks a decisive break with the traditional approach to information management, which is driven by data warehousing and subsequent analytics and is thus essentially *reactive* and plagued by the redundancy and overhead problems described in Background to the Invention.

[0021] Note that in the following, an *observation* is defined as an abstract event tied to a change in enterprise information chain. An observation has a life cycle, which is defined and maintained by the observation management system. Further, an observation can be a composite of other observations. A simple example of an observation is a change in Account Balance of a particular bank customer. The observation generated as a result of such an event will constitute of the following major entities:

- The old and new values of the Account balance for the customer
- The time at which the observation was recorded
- The time for which the observation is valid
- The access rights for the observation

[0022] A key novelty of this approach is that it provides the ability rapidly to deploy observations with minimal intrusion on the existing applications for seamless dynamic integration.

[0023] The proposed platform provides a means to create, catalog, discover, deliver and manage the observations, just as data is managed by database management systems (DBMS). Individual observations in turn can serve as the building blocks of composite observations triggered by the events that affect them.

[0024] The primary objective of the invention is to ease the integration of observing data sources in a coherent manner by providing rapid deployment of observation agents and life-cycle management of observations in a declarative paradigm (a non-procedural, "no-programming-required" paradigm; SQL is an example of a declarative paradigm for data management. The SQL client does not know—or care—how the data is stored, or what its address or location is, or what the database's file structure is; it simply creates and uses data). The platform also provides an observation rules-based technology to filter, transform and compose these observations, to provide *only relevant* knowledge to consuming applications. These rules can be manually or automatically generated depending on the complexity and automation needs of the application.

[0025] The observation management platform will provide the basis for a new class of adaptive enterprise applications based on the observation technology; these applications will rely on enterprise observation integration for just-in-time and correct contextual information for the analytics at hand, rather than on large warehouses. The platform also provides a rules-based observation technology to filter, transform, and compose these observations, and to provide relevant knowledge to the consuming applications.

[0026] (It is important to note that the present invention is not "middleware"—a layer of software between network and applications; rather, it *leverages* middleware. Nor is the present invention a database management system (DBMS); rather, it *uses* DBMS. Nor does it involve integration; rather, it *observes* data at source.) This can be thought of as analogous to a macro program that operates on top of an existing data management application program.

BRIEF DESCRIPTION OF THE DRAWINGS

[0027] Figure 1 is a flow diagram showing a representative enterprise information system of the prior art.

[0028] Figure 2 is a generalized flow diagram showing the present invention.

[0029] Figure 3 is a flow diagram showing the present invention as used in an enterprise information system of the prior art.

[0030] Figure 4 is a detailed diagram of the "observation administration station" shown in Figure 3.

[0031] Figure 5 is a flow diagram representing the use of the invention in an enterprise information system (of the type shown in Figure 1)

[0032] Figure 6a is a screen shot showing the user interface for the Observation Agent Console as it is used to create observations

[0033] Figure 6b is a screen shot showing the user interface for the Observation Agent Console as it is used to configure observations through selection of columns.

[0034] Figure 7a is a screen shot showing the user interface for the Observation Agent Console as it is used to edit observations

[0035] Figure 7b is a screen shot showing the user interface for the Observation Agent Console as it is used to edit observations

[0036] Figure 8 is a screen shot showing the user interface for the Observation Management Console as it is used to monitor the status of an observation

DESCRIPTION OF PREFERRED EMBODIMENTS

[0037] According to one embodiment of the present invention, a system is provided for the leveraging of business data from one or more of a plurality of data sources via an observation management platform. The present invention allows enterprise systems to derive knowledge from data and automate decisions based on knowledge without having to aggregate and integrate data in the conventional way.

[0038] A logical diagram showing the key elements required for the observation agent architecture of the present invention is shown in Figure 2.:

1) **Observation** rules $\mathbf{R_o}$ determine what is to be observed by the observer
2) **Notification** rules $\mathbf{R_n}$ determine what the observer passes on to one or more listeners
3) **Listening** rules $\mathbf{R_L}$ determine what to filter out before passing information on to the analysis engine.

[0039] As shown in FIG 2, the software observer agent (O) observes data events (observations) according to observation rules R_o. The notification component (N) then takes on these observations and puts them in the mailboxes of the listeners according to the notification rules R_n, which relate to mailbox management. (Note that when the term "mailbox" or "mail" is used, the reference is not to e-mails as such, but rather to notification generally.) The listener (L) is a client side component, which after having established a connection to the relevant listener (L) provides the listening rules R_L to the notification agent (N), which is used to customize the observation delivery to the mailbox of the listener (L). After that the listener (L) can access the mailbox in a connected or

disconnected mode as and when required. Typically, each of the software observer agents resides on a server computer with one observer agent associated with each database. In addition, the notify agents also reside on the server computers. Each of the listener agents however resides on a client computer with the client computers and the server computers connected by a communication network (such as intranet or internet).

[0040] Following are the key benefits this platform provides:

- ***Reduced need for wholesale data management and analysis***

[0041] Rather than analyze data in bulk, observation agents simply observe changes in key metrics (also known as "states") and report back on these changes in real time; thus the quantity of data being leveraged never increases

- ***Formal infrastructure to manage pervasive just-in-time enterprise information***

[0042] The platform provides a means to create, modify, catalog, query, deliver and manage observations. An analogy: just as various clients have a consistent view of the data from a DBMS, so the observation management system provides a consistent view of relevant observations from diverse sources to a client.

- ***Rapid Integration of Observed data***

[0043] The platform provides a consistent Application Program Interface (API) to discover and define the data assets to be observed. Once the adapter is provided for a specific data source type, the observation console can declaratively configure and install an observation agent and manage its life- cycle without any programming effort.

- ***Efficient flow of information***

[0044] Since different applications would have different requirements for the observations, a rule based filtering mechanism coupled with composite observing capability would eliminate redundancy and would thus reduce the load on the observed data source. Also these observations can be mixed, matched and transformed via formal semantic rules, to create much higher order of knowledge, which can then be pro-actively delivered to the observing application.

- ***Rule based observations***

[0045] Filtering and transformation rules, both at the observation and listening end would enable applications to dynamically change observation behavior.

- ***Agent-based observations***

[0046] Observation agents will have the characteristics required from software agents of being context sensitive, non-intrusive and proactive. This will be achieved by leveraging on existing infrastructure provided by data sources wherever possible.

- **Integration of data management and analysis with business decision-making in a seamless whole**

[0047] Observation agents observe changes in key metrics; these changes are tied to business decisions via business rules; decisions are thus automated in real time.

- **Self-adaptation via a closed feedback loop**

[0048] The present invention is uniquely self-adapting; customer models are continuously measured against empirical data and improved accordingly; predictive hypotheses are tested and refined—all in real time, via a closed feedback loop.

- **Ease of integration**

[0049] The present invention is a "light footprint" technology, quick and easy to install (unlike current information management systems, which can take years to implement and are in fact never completed).

- **Compatibility with existing information infrastructure**

[0050] The invention adds a layer on top of existing standard platforms like EJB, XML, JDBC, JMS, etc., to provide an integrated solution for observation integration.

[0051] An example of the application of the present invention to the prior art system shown in Figure 1, is shown in Figure 3.

I. OPERATING ENVIRONMENT

[0052] Customer interaction data sources

[0053] In FIG.3 customer 10's and client 1's interaction data sources are a heterogeneous combination of touchpoint-specific, type-specific, platform-specific entities, distributed across the client's enterprise (and usually also geographically). This environment is characterized by:

- Multiple touchpoints (Branch Office 3, Call Center 4, Web Site 5, ATM Machine 6, 9), for both customer information 11 and transactional interaction 12a, 12b
- The various touchpoints are accessed via various mechanisms, both human and machine-based, e.g., walking to branch 3; using the telephone 7 to access call center 4; or using PC 8 to access the Internet
- Each touchpoint is a data source of potentially various types (Relational Database 12b, flat file 12a, Application 12, etc.)
- Each data source type can reside on a particular platform, typically a server computer such as_IBM DB2/Unix, SQLServer/NT, etc.
- The data from multiple touch points (3,4,5,6) are communicated via a communication path (intranet or internet) through observation agents 13a to the central information-processing center 2, where it is stored and possibly "mined" to glean any relevant intelligence, which is then fed to the client's headquarters 1.

[0054] Customer interaction data is interactive and "full duplex," meaning that both informational (web content, promotions, etc) and transactional data (account information, etc.) are put into—and in other cases retrieved from—the data sources. However, significantly more data is put into the systems than is retrieved from them because the act of retrieval is itself recorded and stored. In effect, from a data collection and recording perspective, this interaction is a continuously growing "collection" of data.

[0055] Once the data is stored in the Central Processing Center 2, businesses want to benefit from any intelligence derived from the interaction, so a whole host of applications emerge to facilitate transformation of this continuously collected data into useful business intelligence, usually in a batch mode (nightly, monthly, etc.). This process is loosely

termed "mining," hence the applications are typically called "data mining" applications. The basic function of these applications is to "extract" intelligence from past data. This extraction process is represented as an arrow in Fig 3 from Center 2 to headquarter 1.

[0056] Once mined, the business intelligence uses Business Units 1a and 1b to create or modify business rules that form the basis of various decisions typically distributed across several functional areas of the financial enterprise. This process of "pushing" out the intelligence to the various operations decision points is known as "operationalizing" intelligence. Several applications operate in this context, all "using" this intelligence to positively "impact" the interacting customers and possibly increase the value of future interactions. This is also shown in thick arrows in Fig 3 flowing outward from the headquarter 1.

II. OPERATION

[0057] The invention is a software program comprising a central manager (main program) and several "agent" modules (or sub-programs). In the preferred embodiment, the main program is written in JAVA and executes on any hardware that contains a JAVA. The agent modules are written in JAVA.

- **Observation Agents**

[0058] One Observation Agent may exist for each data source in the enterprise. Thus, for example, an Observation Agent is associated with each server computer containing a database.

- **Notification Agents**

[0059] Only one notification "directory" exists that may have several notification agents within it, one for each observation agent. The notification directory is a brokering and naming service accessed by the observations, the listeners, and an administration console.

- **Listening Agents**

[0060] A listener is a business application specific program that houses the "business application rules" for each business application. One listener exists for each business application that wants to use it.

[0061] FIG-3 shows the physical location and placement of the Observation Platform in an existing enterprise information system. The information adapters 13 in FIG-1 are now replaced, in FIG-3, by the observation agents 13a, and the business enterprise 1 now houses a new "observation administration station" shown as 1c.

[0062] FIG-4 expands upon the details of the "observation administration station" 1c (top left) and information adapters 13a (top right) by showing the key architectural elements of the invention as integrated into an enterprise information system.

[0063] For purposes of explanation, a real-life scenario is provided, involving a representative enterprise application—in this case, a cross selling application in the banking sector, where bank A aims to improve on credit card marketing to its existing checking account customers.

[0064] In the following, it is important to distinguish between 1) business application logic, and 2) Observation Platform Rules (Observation rules, Notification rules and Listening rules). For example, the business application logic for offering a credit card for this purpose is as follows:

> "if the account balance falls below a thousand dollars, and the customer has been there for more than five years, and their fico score is above 550, then offer the credit card in California"

[0065] To support this business application logic, the following observations are relevant:

- Account Balance
- FICO credit score
- Length of membership in California

[0066] The "Observation Platform Rules" allow filtering to achieve this relevancy in observation by observing *only those fields* relevant to the business application served. As shown in FIG 3 (and, at a higher level of abstraction, in FIG-2) the observer (O) observes the data events (observations) according to observation rules R_o, (e.g., an observation rule might be to observe the customer's account balance and FICO credit score for bank customers residing in California). The notification component (N) then takes on these observations and puts them in the mailboxes of the listeners according to the notification rules R_n, which relate to mailbox management. The listener (L) is a client side component, which after having established a connection to the relevant listener (L) provides the listening rules R_L to the notification agent (N), which is used to customize the observation delivery to the mailbox of the listener (L), e.g. notify only when the account balance drops by more than 20% (new:account_balance < old:account_balance * 0.80). After that the listener (L) can access the mailbox in a connected or disconnected mode as and when required.

Observation Agent

[0067] The observation agent (O) 13a is the system component that captures the observation and makes it available to the observation subscribers. The Observation Agent consists of two logical components: an observation component and a notification component. Both components are configurable either via console or via a configuration file (see III. Administration, below). This is primarily a demon-like process, which registers itself with the "Observation Agent Repository" while it boots up and starts performing observations on the data source. The observations are then analyzed and transformed by the notification component and delivered to the mailbox of the subscribers.

[0068] In our example, let there be a customer data (FIG-4, D_1) in an OLTP system for bank called "Virtual Bank of USA". The California office has a requirement that any time the customer cash withdrawals are more than $1000 a day, some specific action needs to be taken. The system administrator creates an observation to observe the account balance which includes the time the transaction happened, the fields to be observed, the time period for which the observation is valid, the access rights for the observation, and

makes the application in California office (FIG-4, A_1) a subscriber to this observation on its request. The notification agent (FIG-4, N_1) would deliver the observations in the mailbox (FIG-4, M_1) of the California office with an added rule that the observations life span is two days, after which, the observations would automatically be cleared off by the notification agent (FIG-4, N_1).

Notification Agent

[0069] The notification agent (FIG-4, N_1) forms the basis for all outbound data observation packets. Logically a separate entity, physically notification agent resides along with the observation agent. The notification agent provides an interface to register listeners. It also provides a mailbox metaphor for messages to be delivered and persisted. It allows for discriminator rules for notification based on observed attributes. The Notification subsystem abstracts out the connectivity for outbound observed data from the observation agent (FIG-4, Ag). The Publishing API (FIG-4, Publishing API) would be the interface used by the notification agent (FIG-4, N_1) to provide the above notifications to the listener (FIG-4, L_1). The publishing API will allow the agent to set up mailboxes (FIG-4, M_1, M_2) for listeners and do the clean up for observations from the mailbox as desired.

[0070] In our example, the publishing APIs would let the observation agent (FIG-4, Ag) set the mailbox (FIG-4, M_1) for the California office (FIG-4, A_1), assign it a lease, and enable the agent to publish the observations to be observed by the listening application (FIG-4, A_1). Following the example, the notification agent would let an observer i.e. the Virtual Bank Application to precisely specify what attributes it is interesting in observing via the notification rules (FIG-2, R_N) and the observation agent (FIG-4, Ag) would set up the mailbox (FIG-4, M_1) for the above application.

Observation API

[0071] The Observation API (FIG-2, Observation API) forms the basis for all in-bound data observation packets. This API abstracts out the connectivity for inbound observed data to the observing agent. Observation API constitutes a consistent interface for a listener to receive the observations by enabling it to consistently connect to the

observation platform. This API would let the listener set the listening rules, e.g. in the example above the application for California could set the rule to listen only to observations related to California.

[0072] In the above example of "Virtual Bank" the API would let the listener rule (FIG-2, R_L) sum(withdrawals) > 1000 on SAME(DAY) NOTIFY and will notify the application at the California office (FIG-4, A_1) of all accounts where the sum of withdrawals exceeded one thousand dollars. Now this application can be a continuously running application whereby anytime a withdrawal is made the above condition is tested, in which case the listener (FIG-4, L_1) acts as a second tier agent, or it can be just a simple batch application run at the end of the day to flag the accounts which satisfy the above condition. In our example, this would mean that the observations can be delivered at the end of the business day, and the application while processing the observation can either decide to do the sum of withdrawals themselves or can delegate it to another agent to flag out accounts where the sum of account balance exceeded one thousand dollar mark.

Management API/Observation Console

[0073] Primarily used by observation console to do agent management. The Observation Management Console (FIG-4, C_1) and other third-party administrative applications use the Management API to configure, manage, discover, and monitor the smooth working of the observation management platform.

[0074] In the example above, the API would let the system integrator, using the console (FIG-4, C_1, which is an application written using the management APIs), to set up the observation if it does not exist; or edit the observation if it does exist but does not serve the purpose of the California office. Once the observation agent has been registered in the name server (FIG-4, Ns_1), the API can keep track of the health of the agent via some heart beat mechanism. The heart beat mechanism can usually be a periodic packet based communication with the name server (FIG-4, Ns_1) to let the system be aware of the health of the agent (FIG-4, Ag).

Listener

[0075] The Listener provides an API to discover, connect and poll observations; a toolbox for authoring discrimination rules (also referred to earlier as business application logic) to filter observed data based on attributes in the data; and an interface to transform observation data into useful business intelligence.

[0076] Listener (FIG-4, L_1) is the logical end point of the agent platform, which provides the adapter 13a for the observing applications (FIG-4, A_1). The listener provides APIs (FIG-4, Listener API) to discover, connect and poll observations. It also provides the discriminator rules (FIG-2, R_L) to filter observed data based on attributes in the data. It also provides an interface to transform observations to suit the purposes of the observing application (FIG-4, A_1). This module constitutes the proxy (a proxy is a pattern frequently used for distributed applications) for the listening application and provides a seamless plug-ability for diverse listening applications. What this implies is that a listening application (FIG-4, A_1) can dynamically decide to use the listening services from the observation platform.

[0077] In the example above the listener (FIG-4, L_1) would be the California office application (FIG-4, A_1), which will use the listening APIs to discover, connect and listen to the required observations. In our example, the customer data (FIG-4, D_1) for "Virtual Bank of USA" is being observed. The California office has a requirement that any time the customer cash withdrawals are more than $1000 a day, some specific action needs to be taken. The system administrator creates an observation to observe the account balance which includes the time the transaction happened, the fields to be observed, the time period for which the observation is valid, the access rights for the observation and makes the application in California office (FIG-4, A_1) a subscriber to this observation on its request. The notification agent (FIG-4, N_1) would deliver the observations in the mailbox (FIG-4, M_1) of the California office with an added rule that the observations life span is two days, after which, the observations would automatically be cleared off by the notification agent (FIG-4, N_1). The notification agent would let an observer (in this case the Virtual Bank Application) know, via the notification rules (FIG-2, R_N), precisely specify what attributes it is interested in observing, and the observation agent (FIG-4, Ag)

would set up the mailbox (FIG-4, M_1) for the above application. These observations can be delivered at the end of the business day, and the application, while processing the observation, can decide either to do the sum of withdrawals themselves or delegate it to another agent to flag out accounts where the sum of account balance exceeded one thousand dollar mark.

Observation Management Console

[0078] The Observation Management Console (FIG-4, C_1) acts as the nerve center for the distributed agents. It will monitor, manage, and co-ordinate the agents. The management console would be used to discover the observable entities (for the UI description of how this will be achieved in the proposed system please see FIG 6a). In the checking account application and data and would provide an observation administrator to choose the account balance as an attribute to be observed (FIG 6a, FIG-6b) along with rules to view only California and Nevada customers (FIG 7b, "Where Clause"). The Console would let the administrator set the quality of observation requirements (FIG-7b, "Poll Interval" and "Rows to Fetch")—e.g., what to observe, and how often, and for how long. Once defined the console would create the required observation information that can then be used by an agent to start an observation.

[0079] In the example the Observation Management Console (FIG-4, C_1) will let the observation administrator discover the data source (FIG-4, D_1), give the administrator the ability to discover the observed attribute "Account Balance" within the data source (FIG-4, D_1) and will then will let the administrator set the observation rules e.g. in this can be the observation rule to observe customers residing in California (state = 'CA') (FIG-2, R_o). Once the information is provided to the observation console (FIG-4, C_1) it will set up the observation agent (FIG-4, Ag), which can then be started automatically or at a later time.

[0080] Once the agent has been up and running, the observation console (FIG-4, C_1) will be used to administer, modify and remove the agents as and when required.

Observation Agent Repository

[0081] This is the naming service for discovery and description of agents. This can be evolved to use Universal Description, Discovery and Integration (UDDI: a universal standard for Web Services model) or some other standard to fit into the overall web services architecture. The observation Agent repository acts as a naming and directory service where the observing applications can query for and get the address for the relevant observation available to the application. The application can then enable listening to the observation.

[0082] The Observation Agent repository in our scenario would hold the observation agent (FIG-4, Ag) names and its observation properties e.g. what is being observed (the attribute list, FIG-6b), how often is it being observed (Polling interval, FIG-7b "Poll Interval"), and what are the conditions to be satisfied before the observation takes place (observation rules, FIG-7b "Where Clause") such that when the California office requires the same observation (FIG-4, Ag) it can send a query to the Observation agent repository (FIG-4, Ns_1) if it can service the request and if it can, request for the agent (FIG-4, Ag) address which could be a URL or a TCP/IP port.

[0083] The naming service provides various mechanisms to search and query the required agents based on its properties. The repository also keeps track of the health status of the agents and can be used by the observation administrator using the Observation management console (FIG-4, C_1) to do observation platform management.

Observing Application

[0084] An observing application (FIG-4, A_1) is an external application which uses the observation management platform to proactively access the events (observations) as and when they occur according to pre-defines quality of service rules defined in the observation platform in the form of observation rules (FIG-2, R_o), notification rules (FIG-2, R_n) and listening rules (FIG-2, R_L). The observing application is provided with an interface in the form of Listener API, which allows it to "plug" into the observation platform. This is accomplished via first discovering the relevant observation agent, making a connection to the agent, notifying it of its listening requirement and getting a

handle for its mailbox (FIG-4, M_1) for subsequent listening.

[0085] In the example provided, the California office application (FIG-4, A_1) is the observing application. The only awareness it has of observable events is the name service (FIG-4, Ns_1). It connects to the name service and sends a query to the name service asking for agent addresses which can provide it customer account balance changes in the Data source (FIG-4, D_1). The name service returns the address of the agent (FIG-4, Ag). The application (FIG-4, A_1) uses this address to connect to the agent, and requests for the required observations. In response the agent notification service (FIG-4, N_1) creates a mailbox for the listening application (FIG-4, A_1) and sends back the address of the mailbox to the application. After which the application start receiving the required observations to be acted upon.

III. ADMINISTRATION

[0086] An example is provided to illustrate the administration process: A loan officer for a financial services company wants to obtain the current credit card balance, APR rate, FICO score and credit limit/line of each customer whenever a customer submits an application for a personal loan. The officer logs in to the system and first finds out (by querying the yellow pages of the naming service directory) if an existing agent is already submitting the above information for the above customer. If so, he creates another listener for the agent for the above information. If not, then he creates an observation that sends his application (or desktop) the above information with a screening clause that triggers this event when the customer's "loan apply" flag is set to 1 (from 0).

- **Data discovery**

[0087] As Figure 5 shows, the user logs into the agent platform using the same administrative console 18 and then queries the registry for a list of all data sources known to the system. He then queries the registry, using the naming service, for a list of all agents currently observing the object of interest. This process is termed “data discovery” and its main purpose is to allow the user to re-use an existing observation already in place or to clone and modify one for a special purpose. This is shown in Fig 7a and Fig 7b. Only when these two cases fail to deliver the data that the user wants does the user either

create a new observation on a registered data source or request the installing administrator to install the platform at another data source location. To view all the observations currently in progress, the user can either search the naming services for key names for observation attributes or list all the observations and select one as shown in Fig. 8.

- **Start from scratch and Create/Activate an observation**

[0088] Once the user has discovered the relevant data, he starts or creates a new observation using an existing data source. Fig 6a and Fig 6b show this console functionality. The observation console informs the user of the various tables available to be observed and the various entities (and their attributes) like field names, field types, field size, etc., that are available to the user to help create and start an observation. Using this console, the user selects the entities of interest to him (that is, to his application) and "creates" an observation by assigning a unique name (conforming to the naming service) and a schedule (real-time, hourly, nightly, etc.). After creating this entity—called the observation—the user then activates it.

- **Clone and Modify to Create/Activate an observation**

[0089] Especially after the observation platform has been in usage for an extended period of time (e.g., several months) enough observation agents are already in place that only slight modifications are usually needed for new applications. So, cloning an existing observation and then modifying the clone is a much more efficient means to create and activate new observations. The steps involved are shown in Fig 7a and Fig 7b, and are identical to the above except preceded by the user cloning an existing observation and using the clone for all subsequent steps (some of these are mentioned in the following section of changing the observation).

- **Edit/Change/Inactivate an observation**

[0090] Typically, due to the growing, evolutionary needs of the application or to fix some problem (or in the case above to create an observation through modification of an existing observation's clone), it is required to modify an existing observation. To do this,

as shown in Fig 7a and Fig 7b, the user logs into the console and selects the observation that needs to be changed. The user first deactivates the observation and then "opens" it to edit. The user than makes any changes to the observation (name, schedule, etc.), saves the changes and then re-activates the observation.